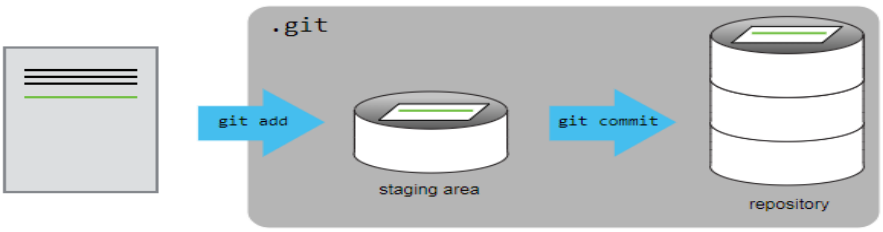
GIT II: Resolució de Conflictes

Staging area

Si penses en Git com prendre instantànies de canvis durant la vida d'un projecte, git add especifica què anirà en una instantània (posant coses a l'staging area), i git commit llavors realment pren la instantània.



Git log

Si volem saber què hem fet recentment, podem demanar a Git que ens mostri la història del projecte usant git log:

$ git log

git log llista tots els commits fets a un repositori en ordre cronològic invers. El llistat de cada commit inclou l'identificador complet del commit (el qual comença amb el mateix caràcter que l'identificador curt que imprimeix l'ordre git commit anterior), l'autor del commit, quan va ser creat, i el missatge de registre que se li va donar a Git quan el commit va ser creat.

Paginació

Quan l'output de git log és massa llarg per cabre a la pantalla, git utilitza un programa per dividir-lo en pàgines de la mida de la pantalla. Notaràs que l'última línia de la teva pantalla és un :, en lloc del teu prompt de sempre.

- Per sortir del paginador, prem q
- Per moure't a la pàgina següent, premeu la barra d'espai.
- Per cercar alguna_paraula a totes les pàgines, escriu /alguna_paraula i navega entre les coincidències prement n (next).

Límit de registres

Per evitar que git log cobreixi tota la pantalla del teu terminal, pots limitar el nombre de commits que Git llista usant -N, on N és el nombre de commits que vols veure. Per exemple, si només voleu informació de l'últim commit, podeu utilitzar:

$git log -1

Git diff

És una bona pràctica revisar sempre els nostres canvis abans de desar-los. Fem això usant git diff. Això ens mostra les diferències entre l'estat actual del fitxer i la versió més recent desada:

$ git diff

De vegades, per exemple en el cas de documents de text, un diff per línies és massa caòtic. És en aquest cas on l'opció --color-words de git diff es torna molt útil ja que ressalta les paraules modificades usant colors

Git status

git status mostra l'estat del directori de treball i l'àrea de l'entorn d'Staging. Permet veure els canvis que s'han preparat, els que no i els arxius on Git no farà el seguiment. El resultat de l'estat no mostra cap informació relativa a l'historial del projecte. Per fer-ho, tal com hem vist has d'usar git log.

Per desfer un git add .

Per desfer un git add abans d'un commit, executeu
git reset <arxiu> o
git reset per desfer tots els canvis.

Resolució conflictes

Usualment, quan gestionem els nostres projectes amb Git podem trobar-nos amb problemes per fusionar canvis que hem realitzat. Per exemple, si tenim diversos desenvolupadors treballant en àrees molt similars, és possible que tinguem algun tipus de conflicte de versions.

Simulem un conflicte

Farem un canvi del mateix fitxer en els dos repositori, en el local i en el remot.
El resultat és que tenim versions diferents del mateix fitxer, per tant el fitxer entrarà en conflicte.

Escenari 1

Si modifiquem el mateix fitxer, però una línia diferent. Al intentar fer un push del fitxer local a remot, obtindrem el següent missatge:

```
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/annaestic/teacher'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Al no modificar la mateixa línia, el git ha resolt el conflicte per si mateix, fent un merge automàtic

```
C:\Users\Anna\Documents\git\annaestic\teacher>git pull
remote: Enumerating objects: 11, done.
remote: Counting objects: 100% (11/11), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 6 (delta 4), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (6/6), 844 bytes | 29.00 KiB/s, done.
From https://github.com/annaestic/teacher
   94fa86a..db87b51  main       -> origin/main
Auto-merging Recursos/CSS/7 fonts/index.html
Merge made by the 'ort' strategy.
 Recursos/CSS/7 fonts/index.html | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Escenari 2

Ara , fem la prova de modificar el mateix fitxer, però de la mateixa línia. Si intentem fer un push del fitxer local a remot, obtindrem el mateix missatge d'abans:

```
To https://github.com/annaestic/teacher
 ! [rejected]        main -> main (non-fast-forward)
error: failed to push some refs to 'https://github.com/annaestic/teacher'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

El que ens diu es que hem d'integrar els canvis, abans de pujar-ho al repositori remot. Si intentem fer un pull, ens segueix donant l'error:

```
C:\Users\Anna\Documents\git\annaestic\teacher>git pull
error: Pulling is not possible because you have unmerged files.
hint: Fix them up in the work tree, and then use 'git add/rm <file>'
hint: as appropriate to mark resolution and make a commit.
fatal: Exiting because of an unresolved conflict.
```

Per solucionar aquest conflicte farem servir el Visual Studio Code i arreglarem el conflicte manualment. Si anem al VS, al fitxer en qüestió, veurem alguna cosa semblant:

```
<body>
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<<< HEAD (Current Change)
    <h1>proves amb fonts e icons local</h1>
=======
    <h1>proves amb fonts e icons remot</h1>
>>>>>>> 38dbc1a0f89bf7908e17a660b383594c721894d9 (Incoming Change)
```

El que hem de fer és netejar el codi i resoldre el conflicte, unificant els canvis manualment.

Un cop resolt podem fer un push com sempre (add + commmit + push)

Escenari 3

Si modifiquem fitxers diferents. Al fer el Git push fa el merge automàticament, incñus sense mostrar cap conflicte:

```
C:\Users\Anna\Documents\git\annaestic\teacher>git pull
remote: Enumerating objects: 11, done.
remote: Counting objects: 100% (11/11), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 6 (delta 4), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (6/6), 846 bytes | 38.00 KiB/s, done.
From https://github.com/annaestic/teacher
   bb638c2..7e32804  main       -> origin/main
Merge made by the 'ort' strategy.
 Recursos/CSS/7 fonts/index.html | 3 +--
 1 file changed, 1 insertion(+), 2 deletions(-)
```

.gitignore

Què passa si tenim fitxers que no volem que Git rastregi.

Per exemple els fitxers dels mòduls que ens genera Bootstrap al crear un projecte, no ens interessa que cada vegada que fem un push GIT ens rastregi si hi ha canvis. Col·locar aquests fitxers sota el control de versions seria un mal ús d'espai en disc. I el que és pitjor, en tenir-los tots llistats, podria distreure'ns dels canvis que realment importen

Així que li direm a Git que els ignori. Ho fem creant un arxiu al directori arrel del nostre projecte anomenat .gitignore, que contingui per exemple:

```
*.dat
results/
node_modules/
```

Aquests patrons diuen a Git que ignori qualsevol fitxer el nom del qual acabi en .dat i tot el que hi hagi al directori results. (OJO Si algun d'aquests fitxers ja estava rastrejant, Git continuarà rastrejant-los.)

Quan hem creat aquest arxiu, la sortida de git status és molt més neta.

Git help

Fes servir la comanda git help per rebre ajuda de les comandes de git en línia.